ОБУЧЕНИЕТО ПО ПРОГРАМИРАНЕ НА НОВАЦИТЕ – ЗАЩО И КАК ?

Теодоси Теодосиев

ШУ “Епископ К.Преславски”, Шумен, teodosi@fmi.shu-bg.net

***Резюме**: Предложеният обзор се опитва да отговори на два въпроса. Трябва ли обучение по програмиране за всички и в каква степен? Вторият въпрос е, как да реализираме това обучение по-резултатно. Разгледани са различни подходи при решаване на този проблем.*

***Ключови думи**: обучение, програмиране, програма, алгоритъм.*

1. Въведение

Заглавието е лека парафраза на статията [14]. Малко разширяваме обсега с обучението на новаците, включвайки и студентите първокурсници. Предложената работа е провокирана от една страна от въпросите (зададени или подсказани) от студентите и от друга от трудностите, с които се сблъскват обучаеми и преподаватели в процеса на обучението по програмиране. Ще направим кратък обзор на повдигнатите в заглавието въпроси и ще мотивираме необходимостта от търсене на отговори.

В днешното информационно общество, овладяването на информационните и комуникационни технологии (ИКТ) е ключова компетентност [15]. Компютрите проникват в техническата инфраструктура на обществото, приемаме за даденост: микровълнови печки, перални машини, автомати за билети, електронни платежни или онлайн системи за резервация за полет и др. Интернет и World Wide Web свързват човечеството.

2. Нужно ли е обучение по програмиране за всички?

През последните 70 години, информационните и комуникационни технологии трансформираха индустриалното в информационно общество. Тази трансформация оказва влияние върху всички аспекти на живота, от икономиката през социалното поведение до науката. Бързото развитие на ИКТ оказва влияние също и върху нашите очаквания от образованието. Има силна потребност информационните и комуникационни технологии да станат част от общото образование във всички фази на образователния цикъл, от начално образование до учене през целия живот.

Налице е също така силна потребност ИКТ да се използват като средство за предоставяне на информацията навсякъде и по всяко време, и така да се

подобри обучението. Фокусираме се върху необходимостта ИКТ образованието да стане част от образованието на различни нива.

Добро общо образование е необходимо, за да се справим с предизвикателството на разбирането на тези бързо променящите се технологии и да бъдем в състояние да се справим с увеличаването на тяхната сложност. Училищата (включително и висшите) са изправени пред трудното решение: кои аспекти на компютърните науки да се опитат да предадат на учениците (студентите) в ограничен период от време? Компютърните науки се състоят от много различни нива, но публиката вижда само техните приложения. Преподавателите могат да бъдат изкушени да се концентрират върху краткотрайни, специфични аспекти на приложенията, да се фокусират върху ниско ниво на умения, като "Как да го направя", а не на разбирането, на тема "Защо и как работи това". И все пак овладяното днес не може да бъде полезно, няколко години по-късно, когато се сблъскаме с нова система или друго приложение. Освен това, преподавателите често се фокусират върху специфични умения за приложението, не само защото тези умения са за незабавна употреба, но също и защото самите те са само потребители на компютрите и не знаят много за това какво се случва "зад екрана".

Училищата, които осигуряват общото образование трябва да насърчават разбирането на основните понятия в компютърните науки. По-късно, по време на работа, рядко има време да се изучават основите. В езикови курсове, например, обучаемите изучават не само лексиката и граматиката, но също така и основни понятия за комуникация. В курсовете по физика се преподават основните закони на природата, като например запазване на енергията, както и техните последици. Тези концепции на общото образование формират основата, която ни помага да се справим с учене през целия живот.

Предлагаме една основна тема на такова образование да бъде идеята на формализация. Целта е да научим обучаемите, че за да направи нещо компютърът, трябва инструкциите да се изразят във формално определена система.

3. Основни понятия в информатиката.

Един от основните аспекти в информатиката е, че всички компютърни системи са контролирани от програми, т.е. чрез строго определени алгоритми, изразени във формална нотация. С цел да се пишат програми, е необходимо да се направи преход от една интуитивна представа за това как един алгоритъм трябва да работи към формална спецификация на актуалния алгоритъм. Съвременното общество делегира все по-нарастващ брой задачи за машините. Тези машини действат като контрольори, които инициират действия, базирани на сегашното им състояние и на получените входни данни. Броят на възможните поведения, предизвикани от различните условия на

околната среда, обикновено е толкова голям, че е невъзможно да се изброят. Въпреки това, ние се стремим, всяко възможно поведение да е "правилно" в някакъв строг смисъл. Начинът, по който можем да направим това е да напишем спецификация, която улавя практическата безкрайност на процесите, които могат да се развиват с течение на времето, в зависимост от получените входни данни. Програмата е такава формална спецификация, и "програма" със сигурност е сред най-фундаменталните концепции, необходими за разбиране на компютрите. Тази идея е една от основните идеи на компютърните науки и следователно част от образование по компютърни науки.

Защо на всички обучаеми трябва да се преподава програмиране? В края на краищата, за почти всички приложения се предоставят инструменти, които са много по-мощни от това, което всеки потребител би могъл да напише и всички ние сме потребители на софтуер, като например текстообработка, електронни таблици, търсачки и др. Ако компютърните потребители вече няма да програмират, не следва ли, че изкуството на програмирането трябва да се преподава само за компютърни специалисти? Ако обучението е само за повишаване на производителността на потребителите с бъдещите офис-приложения, може да се съгласим с тази гледна точка. Но тази гледна точка, въвеждане на компютърните науки като набор от умения, специфични за отделните приложения, има дребен и сериозен недостатък. Дребният недостатък е, че повечето въведения в офис пакетите се концентрират само върху краткотрайни подробности за използване на пакета, което затруднява всеки трансфер на знания към друг пакет. Понятия, които биха могли да бъдат полезни за различните приложения, като например как информацията се кодира и структурира, често се пренебрегват, въпреки че те улесняват овладяването на нов софтуер. Основният недостатък на компютърно обучение, фокусирано само върху умения е по-трудно да се обясни. Нека разгледаме една аналогия с преподаването на математика. Много професии в областта на науката и техниката изискват използването на математически резултати. Учени и инженери са потребители на математиката в същия смисъл, както много хора са потребители на компютри: те проверяват условието (вход) на теорема или формулата и извличат заключенията (изход). По този начин, човек може да твърди, че учени и инженери, само трябва да се научат как се прилагат математически теореми, че понятието "доказателство" е свързано само с професионалните математици. Но вековете опит показват, че всяко математическо образование включва значително време, посветено на доказателства, въпреки че повечето обучаеми едва ли някога ще докажат теорема извън техните математически курсове. Прилагането на математически резултати е въпрос на разбиране, а не само на модел на съпоставяне. Математикът-потребител не е нужно да знае доказателството за всяка формула, а да я използва. Но трябва да развие математическото мислене на базата на концепциите за "теорема" и "доказателство". Също така,

използването на компютри трябва да бъде въпрос на разбиране, а не само на натискане на десен клавиш. Компютърните потребители не трябва да знаят изходния код на всяко приложение, те го използват, но те трябва да имат интуиция за това какво представлява програмата.

Усвоявайки програмирането студентите придобиват чувство на господство над технологичния инструмент и установяват контакт с някои от най-дълбоките идеи в различни сфери, като природни науки, математика и др.

Във времето на готов приложен софтуер, не се нуждаем от програмирането като инструмент, а по-скоро като знание, което ни помага да разберем какво компютърът може и какво не може да направи. Подобно нещо може да се каже за всякакъв вид общо образование. Общото образование рядко се прилага за незабавна употреба, но ни помага да поставим подробностите с преходно значение в по-голяма перспектива.

4. Как да обучаваме по програмиране новаците?

"Защо аз като студент да се науча да програмирам? " Този въпрос, ако му се отговори правилно може да помогне на преподавателя да мотивира студентите, че започвайки да изучават програмиране могат да научат важни за бъдещата им реализация неща. Програмирането включва в себе си способност за генериране на решения на проблеми. Пораждащото решение означава, че един от резултатите от обучението се явява умението за решаване на проблеми и освен това, ако проблемът е сложен, може да се разбие на подпроблеми и накрая да се даде обобщено решение. Ето защо, на студентите най-напред им се налага да намерят решение на проблема, а след това да се замислят как да комуникира тяхното решение с компютъра, използвайки синтаксис и граматика чрез строг начин на мислене [11,19]. Последното допринася за естествените езикови умения на студентите, защото те са длъжни да се научат да се изразяват недвусмислено ако искат компютърът-неинтелигентна машина да извърши това, което са задали [5].

Умението за решаване на проблеми може да се разшири за решаване на "реални" задачи в различни области с изчислителни цели. Възможността за предаване на тези и други навици се явява аргумент за Feurzeig [3] и неговите колеги за въвеждане на програмирането като способ, помагащ на студентите да разберат математическите понятия като строго мислене, променлива, функция, декомпозиция, проверка и обобщение.

Обикновено информатиката се определя като дисциплина от ново поколение, защото е свързана с математика, физика, техника, лингвистика, философия, психология, икономика, обществените науки в цялост. Ако от една страна тази сложност ни води към относително трудна задача за изследователите в тази област, от друга страна може да се разчита на интересните резултати вече получени в горе изброените дисциплини.

Педагогически контекст на знанията (ПКЗ) се определя като знание, което позволява на преподавателя да превърне своите знания по предмета в нещо достъпно за своите ученици. В аспекта на ПКЗ се концентрираме на необходимостта учителя да представи и формулира темата, така че разбирането да се осъществи. Педагогическото съдържание на знанието включва разбирането какво прави усвояването на специфични теми леко или трудно. Всички изследователи в тази област се съгласяват с твърдението, че ПКЗ е знание, което се развива с годините преподавателски опит.

4.1. Базови елементи в обучението.

Програмирането е едно от основните умения, които трябва да бъдат усвоени от студентите по CS в края на първата година от тяхното следване. Очаква се също така, те да знаят основните концепции на програмирането и да са развили алгоритмичния подход към решаване на проблеми. Сама по себе си тази задача е твърде сложна [4, 20] и често довежда начинаещите до недоразумения и заблуди. Основни понятия за програмиране включващи знанието на контролните структури (последователност, селекция и итерация), механизми за агрегиране (масив, структура, съюз), указатели, функции и др. [13] се усвояват паралелно с усвояване на формален език за програмиране. Когато преподаваме програмиране, нашата цел е да научим начинаещите студенти на основните принципи за използване на програмните концепции, независимо от езика за програмиране. Трудностите са свързани с дуалността в гледната точка към езика за програмиране: от една страна като инструментално средство за формално описание на алгоритмите, а от друга като обект за изучаване. Всички тези трудности водят при много обучаеми до затрудняване на усвояването на материята и до незаинтересованост от програмирането.

Weigend [21] установява, че дори при правилни разсъждения или намерено практическо решение на проблем, новаците се затрудняват да напишат правилна програма. Причината е в невъзможността за преход от мисловната интуиция към комуникативна форма или казано по друг начин в семантиката на програмата. Семантиката е също проблемен аспект на програмирането. Това е така, защото изисква от студента да сглоби различните части на програмата (променливи, изрази, оператори, управляващи структури, обекти, методи) в работеща цялост. Семантиката е тясно свързана и с дейността по тестването и правилността на програмата. Когато се предлага език за програмиране с по-сложен синтаксис обучаемите се сблъскват със синтактични и семантични проблеми.

Linn и Dalbey [8] предлагат "идеалната" последователност за обучение по програмиране, която постепенно преминава от разбиране на програмата към генериране на програма. Тя се състои от три основни звена: особености на

езика за програмиране, навици за проектиране и умения за решение на проблеми. Според Linn и Dalbey "идеалната" верига започва с разбиране на възможностите на езика, знания, които могат да бъдат оценени, чрез искане студентите да преформулират или изменят малко програмата. Второто звено се състои от навиците за проектиране, които се отнасят към групата на методите, използвани за съвместяване на възможностите на езика с формиране на програмата. Това звено на веригата включва и шаблони (стереотипни кодови шаблони, където има повече от една функция) и процедурни умения (използват се за комбинирани шаблони и особености на езика за решение на нови задачи, включващи планиране, тестване и преформатиране). Третото звено на веригата, умения за решаване на проблеми е полезно за изучаване на нови формални системи. Уменията за решаване на проблеми може да се оцени като поискаме от студентите да решат проблем, използвайки непозната формална система като непознат език за програмиране. Въпреки, че тази верига на когнитивни достижения изисква много време, тя е добро резюме на това какво се разбира под задълбочено изучаване на "Увод в програмирането".

В програмирането можем да разграничим два вида знания, а именно създаване на програмата и разбирането на програма [9]. В първия случай, програмистът анализира проблема, създава алгоритмично решение, и след това превежда този алгоритъм в програмен код. Това означава, че студентите трябва да се тренират в процес за решаване на проблеми, размисъл върху този процес и в развитието на алгоритмично мислене [3,5]. Що се отнася до разбиране на програмата, от програмиста се иска да демонстрира своето разбиране, за това, как работи дадена програма.

Govender [4] определя от техническа гледна точка, три основни аспекта, които студентите трябва да научат: данни, инструкции и синтаксис. Към данните се отнасят понятията: променлива и тип на данните. Що се отнася до инструкциите необходимо е разбиране на управляващите структури и подпрограмите. Синтаксисът обозначава група от правила, определящи, какво е разрешено, а какво не в езика за програмиране. Синтактичните правила определят как може да се построят програмите, използвайки методи като цикли, разклонения и подпрограми.

Когато започвате работа като начинаещ програмист или пробвате да четете код на нов език за програмиране всичко ви изглежда еднакво непостижимо. Докато не разберете непосредствено езика за програмиране, вие не можете да видите очевидните синтактични грешки. В течение на първата фаза на изучаване вие започвате да разпознавате неща, които обикновено се наричат “стил на кодиране”. Започвате да забелязвате код, който не съответства на строгите стандарти. Доколкото набирате все по-голям опит в работата в конкретна среда на разработка, се учите да виждате и други неща. Неща, които могат да са съвсем верни от гледна точка на използвания

език за програмиране и абсолютно съответни на договорките за кодиране, но които ни карат да сме нащрек.

Преодолявайки тези грешки стигаме до по-сериозен проблем. От гледна точка студент-компютър Реа [12] определя съществуването на устойчиви концептуално независими от езика за програмиране "грешки" в програмирането и разбирането на програмите от новаците. Отправна точка на анализа на концептуалните "грешки" е в това, че обучаемите искат да общуват с компютъра като с човек, той да интерпретира техните инструкции (това се счита като Superbug). Реа отделя три различни вида концептуални "грешки". Първата е "паралелизъм" – грешка отнасяща се към предположението, че различните клонове на програмата могат да са активни едновременно. Друга грешка е "наивността", с която обучаемите считат, че компютрите излизат от рамките на предоставената информация в рамките на програмния код при изпълнение на програмата. Много важна концептуална грешка на новаците е "егоцентризма", те считат, че това което искат да получат е по-важно от това, което са написали в кода на програмата. (Например, "Не печатай това, което говоря, а печатай, това което мисля").

4.2. Проблеми на преподаването.

При преподаване на програмиране сме изправени пред някои проблеми, присъщи на самото преподаване, както и свързани с организацията на учебния процес. Студентите са склонни да придобиват някои лоши навици за програмиране в опита си да издържат изпита. Има случаи (за съжаление в последно време се увеличават), в които студентите научават големи фрагменти програмен код наизуст, без или с трудно разбиране. Има и студенти, които пишат дълъг програмен код, без какъвто и да е синтаксис и логически тестове, който предизвиква огромен брой грешки, и обезсърчава студентите от програмирането като цяло.

Студентите обявяват липсата на интерес и отвличане на вниманието от други задължения в обучението си за най-честите проблеми [13]. Въпреки това, някои от най-тежките проблеми, като например страх от програмирането, са най-вече в корелация с липсата на практика по програмиране, както и неразбиране на основните концепции за програмиране и синтаксиса на езика за програмиране.

Има няколко проблема, с които трябва да се справим в процеса на обучението по програмиране. Най-важният от тях е огромната диспропорция в предварителната подготовка на студентите. Програмирането е задължителен основен курс за студентите през първата учебна година. Входните им познания и техники силно зависят от предишното им образование. За много от тях, разбира се, това е тяхната първа среща с програмирането, така че те нямат никакъв опит. От друга страна, има някои ученици с умерен, или дори добър

опит в програмирането. Необходимо е да има дисциплина за изравняване на нивото на студентите, колкото е възможно. Такава стъпка вече е реализирана в Шуменския университет като факултативна (за студентите от специалност Информатика). Подобна стъпка е направена и в университета във Вараждин, Хърватия. Там до това решение се е стигнало като този курс е бил факултативен, но при посещение от 80% от студентите в първи курс е станал задължителен.

Непрекъснато обучение играе основна роля в хода за курса по програмиране. Причината за това е фактът, че разбирането на следващите понятия, в които се обучават до голяма степен зависи от разбирането на тези, които преди това се преподават.

Заключение

В работата са цитирани автори от различни страни (Финландия, Холандия, Германия, Швейцария, Хърватска, САЩ и др.), което показва, че тези въпроси са актуални и извън нашата страна. Ако приемем за положителен отговорът на въпроса, поставен в точка 2, то още по-важно става как да обучаваме. Основен проблем е повишаване на мотивацията и облекчаване на материята.

Личният ми преподавателски опит, а и този на други колеги [18] показва, че при програмирането правилното начално обучение играе съществена роля за ефективността и качеството на дейността. Това предизвиква много преподаватели да търсят нови (различни от стандартните) подходи в началното обучение по програмиране. Например:

- Проблемно-ориентирано обучение в група [7]. Студентите казват, че харесват социалните аспекти от обучението в група. Обикновено студентите ценят активното си участие в курса. От друга страна груповата динамика на трудностите, изискването за изучаване на навиците на другите предизвиква проблеми, които трудно се преодоляват от някои студенти.
- Софтуерни инструменти за подпомагане на студентите да придобиват полезно умения за програмиране, както и подобряване на процеса на преподаване на програмиране на начинаещи [13].
- Използването на подходящи примери е ключов, за да се учат абстрактни, теоретични концепции. Интерактивен компютърен софтуер позволява да използваме такива примери, за да създадат привлекателна учебна среда, която не само да се покаже пред обучаемите, но също така да увеличи трансфера на знания [1].
- В Германия използват възрастово-ориентираното обучение и специализирани програмни среди [6]. Въпреки, че дизайнът на тези среди е възрастово-ориентиран, тези които работят с тях за решаване на алгоритмични проблеми, често имат проблеми в класната стая. Тези

инструменти дават обратна връзка на учащите се, на базата на анализ на текущия опит за решение без да се взема предвид предходния процес на решаване на проблемите.

Има и други подходи: с акцент на графиката [2], използване на програмируеми мобилни роботи [10], обектите в началото [17] и др.

Притеснителното е, че у нас се увеличава броят на студентите дори от специалности "Математика и информатика" и "Информатика", които имат нужда от отговор на въпроса "защо да учим програмиране?" Въпреки, че в средното училище те са учили информатика (в частност модулите "Алгоритми" и "Увод в програмирането"), голяма част от студентите, по мои наблюдения нямат сериозни познания нито по съставяне на алгоритъм, нито по неговото формално описание. Това е причината курсът по "Увод в програмирането" в университетите в България да започва отначало. Следователно, за да могат да създават компютърни програми, успоредно с усвояването на език за програмиране, студентите трябва да се учат и на принципите на програмирането.

Благодарности

Работата е финансирана по проект РД-05-280/15.03.2012 на Шуменския университет.

Литература

1. Arnold, R., Langheinrich, M., Hartmann, W. InfoTraffic: teaching important concepts of computer science and math through real-world examples, SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education, ACM New York, NY, USA, 2007.
2. Djordjevic, M. Teaching Introductory Programming Course with Progressive Graphics Examples. Proceedings of the 2007 Computer Science and IT Education Conference, Mauritius, 2007, pp. 177-185.
3. Feurzeig, W., Papert, S., Bloom, M., Grant, R., Solomon, C. Programming-language as a conceptual framework for teaching mathematics. *Newsletter SIGCUE Outlook*, 4(2), 13–17, 1970.
4. Govender, I. Learning to Program, Learning to Teach Programming: Pre- and In-service Teachers'Experiences of an Object-oriented Language. University of South Africa, 2006.
5. Hromković, J. Contributing to general education by teaching informatics. In: Mittermeir, R.T. (Ed.), ISSEP 2006, LNCS, 4226, 25–37.
6. Kiesmüller, U.; Brinda, T.: How do 7th graders solve algorithmic problems?: a tool-based analysis Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE, 2008, art. no. 1384395, p. 353
7. Kinnunen, P., Malmi, L. Problems in Problem-Based Learning –Experiences, Analysis and Lessons Learned on an Introductory Programming Course, Informatics in Education, 2005,Vol. 4, No. 2, 193–214

8. Linn, M.C., Dalbey, J. Cognitive consequences of programming instruction. In: Soloway, E., Spohrer, J.C. (Eds.), *Studying the Novice Programmer*. London, Lawrence Erlbaum Associates, 1989, 58–62.
9. Mannila, L. Novices' Progress in Introductory Programming Courses, Informatics in Education, 2007, Vol. 6, No. 1, 139–152
10. Pásztor, A., Pap-Szigeti, R., Lakatos Török, E. Effects of Using Model Robots in the Education of Programming. Informatics in Education, 2010, Vol. 9, No. 1, 133–140
11. Papert, S. Mindstorms. Children, Computers and Powerful Ideas. Basic Books, Inc. Publishers, NewYork, 1980.
12. Pea, R.D. Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 1986, 2, 25–36.
13. Radošević, D., Orehovački, T., Lovrenčić, A. Verificator: Educational Tool for Learning Programming, Informatics in Education, 2009, Vol. 8, No. 2, 261–280
14. Reichert, R., Nievergelt, J., Hartmann, W. Programming in schools – why, and how? In C. Pellegrini,A. Jacquesson (Hrsg.): Enseigner l'informatique, 2001, pp 143-152. Georg Editeur Verlag.
15. Reichert, R. *Theory of Computation as a Vehicle for Teaching Fundamental Concepts of Computer Science*. Doctoral Thesis. No. 15035. ETH Zurich, 2003 http://e-collection.ethbib.ethz.ch/show?type=diss&nr=15035
16. Saeli, M., Perrenet, J., Jochems, W., Zwaneveld, B. Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective Informatics in Education, 2011, Vol. 10, No. 1, 73–88
17. Sajaniemi, J., Hu, C. Teaching Programming: Going beyond "Objects First". University of Joensuu, Department of Computer Science, Technical Report, Series A, Report A-2006-1, http://www.cs.joensuu.fi/~saja/publications.html.
18. Skupiene J. Programming Style – Part of Grading Scheme in Informatics Olympiads: Lithuanian Experience, ISSEP, 2006, p. 545-552
19. Szlávi, P. Zsakó, L. Programming versus application. In: Mittermeir, R.T. (Ed.), ISSEP 2006, LNCS4226, 48–58.
20. Van Diepen, N. Elf redenen waarom programmeren zo moeilijk is (in English: Eleven reasons why programming is so difficult),Tinfon, 2005, 14, 105–107.
21. Weigend, M. From intuition to programme. Programming versus application. In:Mittermeir, R.T. (Ed.), *ISSEP 2006, LNCS*, 4226, 117–126

TRAINING IN PROGRAMMING FOR NOVICES – WHY AND HOW?

Teodosi Teodosiev

Abstract: *The proposed review attempts to answer two questions. Is training program for all and to what extent. The second question is how to realize this training more effective. Are considered different approaches to solving this problem.*